การเขียนโปรแกรมสำหรับการคำนวณแบบขนาน ตอนที่ 2 การเขียนโปรแกรมขนาน

จินต์-นุดี วรมหาภูติ

ในฉบับที่ 39 ผู้เขียนได้แนะนำระบบคอมพิวเตอร์แบบหลายโปรเซสเซอร์แบบต่าง ๆ คอมพิวเตอร์แบบใหม่นี้มีความสามารถสูงกว่าคอมพิวเตอร์แบบโปรเซสเซอร์เดี่ยวมาก แต่ความสามารถนี้จะไร้ประโยชน์ หากปราศจากโปรแกรมที่สามารถคำนวณแบบขนานได้อย่างมีประสิทธิภาพ สไตล์การเขียนโปรแกรมประเภทนี้ขึ้นอยู่กับโครงสร้างของคอมพิวเตอร์ที่จะใช้งาน อัลกอริทึมบางอันจึงสามารถทำงานได้ดีเฉพาะกับบางโครงสร้างเท่านั้น ในฉบับนี้ผู้เขียนจะแนะนำเครื่องมือที่จำเป็นในการเขียนโปรแกรมเพื่อคำนวณแบบขนานสำหรับเครื่องระบบต่าง ๆ ดังที่เกริ่นไว้ในฉบับที่ 39 ว่า เทคนิคการเขียนโปรแกรมสำหรับเครื่องระบบผสมนั้นเหมือนกับระบบหน่วยความจำร่วม จะผิดกันในแง่ที่เราพิจารณาถึงประสิทธิภาพของอัลกอริทึมของเครื่องระบบผสม ซึ่งเปลี่ยนไปตามการวางตำแหน่งข้อมูล แต่เราจะไม่พูดไปลึกถึงขนาดนั้น ดังนั้นผู้เขียนจะเสนอเพียงเทคนิคการเขียนโปรแกรม และเครื่องมือที่จำเป็นสำหรับระบบหน่วยความจำร่วมและระบบส่งผ่านข้อมูลเท่านั้น

1. ระบบหน่วยความจำร่วม (shared-memory)

การเขียนโปรแกรมเพื่อการคำนวณแบบขนานสำหรับเครื่องระบบนี้ค่อนข้างจะยุ่งยาก เราจึงต้องเพิ่มเติมและดัดแปลงภาษาที่จะใช้สำหรับการเขียนโปรแกรมเพื่อการคำนวณแบบขนานสำหรับเครื่องประเภทนี้เป็นพิเศษ จุดใหญ่ ๆ ที่จำเป็นจะต้องเพิ่มเติมคือ

ก. คำสั่งในการแบ่งแยกระหว่างข้อมูลที่เป็นส่วนตัวของแต่ละโปรเซสเซอร์กับข้อมูลกลาง ซึ่งทุกโปรเซสเซอร์สามารถใช้งานได้

ข. คำสั่งในการแบ่งแยกระหว่างส่วนของโปรแกรมที่คำนวณแบบอนุกรมกับส่วนของโปรแกรมที่คำนวณแบบขนาน

ค. คำสั่งในการควบคุมการใช้งานของข้อมูลกลางให้เป็นไปอย่างถูกต้องตามลำดับ เนื่องจากทุกโปรเซสเซอร์สามารถใช้งานข้อมูลกลางได้ ถ้าไม่มีการควบคุมที่ดีพอ จะเกิดการแทรกแซงซึ่งกันและกัน และจะมีผลให้ข้อมูลกลางผิดพลาดได้

ง. คำสั่งในการควบคุมให้โปรเซสเซอร์หลาย ๆ ตัวทำงานอย่างสอดคล้องกัน (synchronization)

ก่อนจะกล่าวต่อไป ผู้เขียนจะขอนิยามคำว่า โปรเซส (process) โปรเซสคือโปรแกรมหรือส่วนของโปรแกรมที่กำลังคำนวณอยู่บนโปรเซสเซอร์ตัวใดตัวหนึ่ง ดังนั้นในขณะที่โปรแกรมกำลังคำนวณแบบอนุกรม โปรแกรมจะประกอบด้วยหนึ่งโปรเซส แต่ถ้าโปรแกรมกำลังคำนวณแบบขนานอยู่ โปรแกรมนั้นจะประกอบด้วยหลายโปรเซส ซึ่งแต่ละโปรเซสคำนวณอยู่บนโปรเซสเซอร์หนึ่งตัว

หมายเหตุ คำนิยามของโปรเซสข้างต้นต่างไปจากนิยามของโปรเซสโดยทั่วไปเล็กน้อย เพราะผู้เขียนต้องการให้ง่ายต่อความเข้าใจ

1.1 การแบ่งแยกระหว่างข้อมูลส่วนตัวกับข้อมูลส่วนกลาง

ในการเขียนโปรแกรมสำหรับระบบหน่วยความจำร่วมนั้น สิ่งแรกที่จำเป็นที่สุด คือ การบอกความแตกต่างระหว่างข้อมูลส่วนตัวกับข้อมูลส่วนกลาง ในปัจจุบันยังไม่มีวิธีการที่เป็นมาตรฐานที่แน่นอนสำหรับการบอกความแตกต่างของข้อมูลดังกล่าว วิธีที่

```
program   sample;
shared    A, B, C,   : integer;
var       D          : integer;
```

รูปที่ 5 การกำหนดข้อมูลส่วนตัวกับข้อมูลส่วนกลาง

ใช้ขึ้นอยู่กับภาษาที่ใช้เขียน และผู้ออกแบบภาษานั้น ๆ แต่ความแตกต่างเหล่านี้ไม่เป็นอุปสรรคต่อการเขียนโปรแกรมหรือการดัดแปลงโปรแกรมเพื่อไปใช้งานบนเครื่องรุ่นอื่นมากนัก

ในตัวอย่างของบทความชุดนี้ เราจะแบ่งแยกระหว่างข้อมูลส่วนตัวกับข้อมูลส่วนกลางในช่วงการกำหนดประเภทของตัวแปรต่าง ๆ (declaration section) ดังตัวอย่างในรูปที่ 5 ตามตัวอย่างดังกล่าว ตัวแปร A B และ C ถูกกำหนดให้เป็นข้อมูลร่วมในขณะที่ตัวแปร D ถูกกำหนดให้เป็นข้อมูลส่วนตัว

ความแตกต่างระหว่างข้อมูลส่วนตัวกับข้อมูลส่วนกลางคือ ตัวแปรที่เป็นข้อมูลส่วนตัวของแต่ละโปรเซสจะถูกเก็บอยู่ในหน่วยความจำที่ต่างกัน การใช้งานตัวแปรนี้โดยโปรเซสหนึ่งจะไม่มีผลกระทบต่อตัวแปรชื่อเดียวกันของโปรเซสตัวอื่น ๆ ส่วนตัวแปรที่เป็นข้อมูลส่วนกลางนั้น จะถูกเก็บในหน่วยความจำร่วม การใช้งานตัวแปรนี้โดยโปรเซสหนึ่ง จะมีผลกระทบต่อตัวแปรชื่อเดียวกันของโปรเซสตัวอื่น ๆ เนื่องจากตัวแปรชื่อเดียวกันของแต่ละโปรเซสจะถูกเก็บอยู่ที่หน่วยความจำที่เดียวกัน

1.2 คำสั่งแสดงการคำนวณแบบขนาน

เช่นเดียวกับการแบ่งแยกระหว่างประเภทของตัวแปรในข้อ 1.1 ในปัจจุบันยังไม่มีมาตรฐานที่แน่นอนในการแบ่งแยกระหว่างส่วนของโปรแกรมที่ถูกคำนวณแบบอนุกรมกับแบบขนาน เพื่อให้ง่ายต่อความเข้าใจ เราจะใช้คำสั่ง PARBEGIN และ PAREND ซึ่งเป็นที่นิยมใช้กันแพร่หลายในการอธิบายอัลกอริทึมของโปรแกรมที่คำนวณแบบขนาน

PARBEGIN เป็นจุดเริ่มต้นที่การคำนวณของโปรแกรมเปลี่ยนจากการคำนวณแบบอนุกรมโดยหนึ่งโปรเซสเซอร์ มาเป็นการคำนวณแบบขนานที่กระทำโดยหลายโปรเซสเซอร์พร้อม ๆ กัน

PAREND เป็นจุดที่การคำนวณแบบขนานสิ้นสุดลง โปรแกรมส่วนที่ตามมาจะถูกคำนวณแบบอนุกรม

ตัวอย่างในรูปที่ 6 แสดงการคำนวณหาค่าผลบวกของกำลังสองของตัวแปร A B และ C เราจะเห็นได้ว่าการคำนวณหาค่ากำลังสองของ A B หรือ C สามารถทำได้โดยอิสระไม่ขึ้นต่อกัน ดังนั้นเราจึงสั่งให้การคำนวณส่วนนี้ทำแบบขนานโดยโปรเซสเซอร์ 3 ตัว (สามโปรเซส) พร้อม ๆ กัน หลังจากนั้น การหาผลบวกของค่ายกกำลังทั้งสามสามารถคำนวณได้โดยโปรเซสเซอร์เพียงตัวเดียว

```
PARBEGIN
        A := A ** 2;
        B := B ** 2;
        C := C ** 2;
PAREND
D := A + B + C;
```

รูปที่ 6 ตัวอย่างการใช้งานคำสั่ง PARBEGIN/PAREND

1.3 การควบคุมการใช้ข้อมูลกลาง

ปัญหาที่สำคัญที่สุดปัญหาหนึ่งในการเขียนโปรแกรม เพื่อการคำนวณแบบขนานสำหรับระบบหน่วยความจำร่วม คือ **การควบคุมการใช้งานข้อมูลกลาง** เพื่อให้เห็นปัญหาอย่างชัดแจ้ง ผู้เขียนจะขอยกตัวอย่างให้ดูดังรูปที่ 7 ในตัวอย่างนี้ ระบบคอมพิวเตอร์ประกอบด้วยจอเทอร์มินอล 2 ตัว ใช้สำหรับนับจำนวนลูกค้าที่เข้ามาในร้านทั้งทางประตูหน้าและประตูหลังรวมกัน ทุกครั้งที่ลูกค้าเดินเข้ามาในร้าน พนักงานเปิดประตูจะกดปุ่ม Return ให้คอมพิวเตอร์นับจำนวนลูกค้าเพิ่มขึ้นอีกหนึ่ง เมื่อเราพิจารณาดูคำสั่งในโปรแกรมแล้ว ก็ไม่เห็นว่าน่าจะมีอะไรผิด เพราะทุกครั้งที่กดปุ่ม Return ตัวแปร Counter จะถูกบวกเพิ่มอีกหนึ่ง แต่ตามความเป็น

```
program     simple;
shared      counter : integer ;
procedure   CUSTOMER;
    begin
        while true do
        begin
            read (input);
            counter := counter + 1;
        end;
    end;
begin
    PARBEGIN
        call CUSTOMER;
        call CUSTOMER;
    PAREND
end.
```

รูปที่ 7 ตัวอย่างโปรแกรมที่ใช้ข้อมูลร่วมโดยไม่มีการควบคุม

จริงโปรแกรมจะทำงานผิดพลาดถ้าปุ่ม Return ของจอเทอร์มินอลทั้งสองถูกกดลงพร้อมกัน ทำให้การคำนวณคำสั่ง counter := counter + 1 ของโปรเซสทั้งสองแทรกแซงซึ่งกันและกัน ถ้าเราแปลงคำสั่งดังกล่าวเป็นภาษาแอสเซมบลีแล้ว จะเป็นดังรูปที่ 8

```
LOAD    Counter
ADD     1
STORE   Counter
```

รูปที่ 8 แอสเซมบลีของ Counter := Counter + 1

สมมติให้ค่า Counter ปัจจุบันเป็น 10 เมื่อคนเป้าประตูทั้งสองกดปุ่ม Return ในเวลาพร้อมกัน (สมมติให้คนแรกกดก่อนเสี้ยววินาที) โปรเซส 1 จะโหลดค่า Counter ซึ่งเท่ากับ 10 บวกด้วยหนึ่ง และเขียนผลลัพธ์ 11 กลับไปยังหน่วยความจำ แต่ก่อนที่โปรเซส 1 จะบวกหรือเขียนผลลัพธ์เสร็จ โปรเซส 2 ก็โหลดค่า Counter ซึ่งยังคงเท่ากับ 10 อยู่ มาบวกด้วยหนึ่งและเขียนผลลัพธ์กลับไปยังหน่วยความจำในเวลาใกล้เคียงกัน ผลลัพธ์ของโปรเซส 2 จึงทับผลลัพธ์ของโปรเซส 1 ทำให้ค่าใน Counter ใหม่ 11 ทั้ง ๆ ที่ถูกควรจะเป็น 12

จากตัวอย่างข้างต้น เราจะเห็นว่าโปรแกรมทำงานผิดพลาด เมื่อหลายโปรเซสเซอร์พยายามใช้ข้อมูลร่วม ซึ่งในตัวอย่างคือ Counter ปัญหานี้แก้ไขได้โดยให้โปรเซสหนึ่งเพิ่มค่าของข้อมูลร่วม อีกโปรเซสหนึ่งจะต้องรอให้โปรเซสแรกทำงานเสร็จก่อนถึงจะทำงานของตัวเองได้ **เทคนิคการแก้ปัญหาการใช้ข้อมูลร่วมนี้เรียกว่า "การให้สิทธิพิเศษ" (Mutual exclusion)**

การให้สิทธิพิเศษต่อโปรเซส จะจำเป็นต่อเมื่อโปรเซสต้องการใช้งานข้อมูลร่วม หากโปรเซสใด ๆ ทำการคำนวณที่ไม่เกี่ยวกับข้อมูลร่วมก็ไม่จำเป็นต้องให้สิทธิพิเศษ เมื่อโปรเซสหนึ่งกำลังใช้ข้อมูลร่วม เราเรียกเหตุการณ์ช่วงนี้ว่าโปรเซสอยู่ในแดนวิกฤต (Critical section) จากตัวอย่างที่ผ่านมาเราจะเห็นได้ว่า เมื่อโปรเซสหนึ่งเข้าสู่แดนวิกฤต โปรเซสตัวอื่น ๆ จำเป็นต้องอยู่นอกแดนวิกฤต เพื่อให้โปรแกรมทำงานได้ถูกต้อง ในขณะที่โปรเซสหนึ่งกำลังอยู่ในแดนวิกฤต โปรเซสอื่น ๆ ซึ่งกำลังคำนวณที่ไม่ยุ่งเกี่ยวกับข้อมูลร่วม (ไม่ได้อยู่ในแดนวิกฤตของโปรเซสนั้น) สามารถจะคำนวณต่อไปได้อย่างไม่มีปัญหา แต่ถ้าโปรเซสใดต้องการเข้าสู่แดนวิกฤตของตัวเอง โปรเซสนั้นจำเป็นต้องหยุดรอให้โปรเซสที่อยู่ในแดนวิกฤตทำงานเสร็จและออกจากแดนวิกฤตเสียก่อนถึงจะเข้าไปในแดนวิกฤตได้ พูดง่าย ๆ คือ การให้สิทธิพิเศษ หมายถึง การควบคุมให้โปรเซสเข้าสู่แดนวิกฤตได้ทีละหนึ่งโปรเซสเท่านั้น **การคำนวณในแดนวิกฤตจำเป็นต้องทำให้เร็วที่สุดเท่าที่จะเป็นไปได้** เนื่องจากอาจจะมีหลายโปรเซสรอเข้าไปทำงานในแดนวิกฤตของโปรเซสนั้นบ้าง การเขียนโปรแกรมส่วนที่เป็นแดนวิกฤตจำเป็นต้องทำด้วยความระมัดระวังอย่างยิ่ง เพราะหากมีโปรเซสใดเกิดเข้าไปคำนวณแบบไม่รู้จบอยู่ในแดนวิกฤต โปรเซสอื่น ๆ ก็จะไม่สามารถทำงานต่อไปได้เช่นกัน

การให้สิทธิพิเศษต่อโปรเซสหนึ่งในการใช้งานข้อมูลร่วมเป็นปัญหาที่ใหญ่ที่สุดปัญหาหนึ่งในการเขียนโปรแกรมเพื่อการคำนวณแบบขนาน นักวิจัยได้พยายามค้นหาวิธีที่จะให้การควบคุมการให้สิทธิพิเศษเป็นไปอย่างมีประสิทธิภาพ วิธีการทั้งทางซอฟต์แวร์และทางฮาร์ดแวร์ถูกเสนอขึ้นมาหลายวิธี บางวิธีอยู่ในระดับสูง (เขียนด้วยโปรแกรมทั่ว ๆ ไป) บางวิธีก็อยู่ในระดับต่ำ (ระดับระบบควบคุม) ส่วนใหญ่วิธีทางซอฟต์แวร์จะเป็นที่นิยมมากกว่าทางฮาร์ดแวร์ เพราะวิธีทางซอฟต์แวร์มักไม่ขึ้นอยู่กับโครงสร้างของคอมพิวเตอร์ (ในแง่ของโปรแกรมเมอร์) โปรแกรมที่เขียนบนเครื่องรุ่นหนึ่ง สามารถนำไปคำนวณบนเครื่องอีกรุ่นหนึ่งได้ง่าย ถ้าใช้วิธีทางฮาร์ดแวร์สไตล์การเขียนจะถูกจำกัดเฉพาะเครื่องรุ่นหนึ่ง ทำให้ยากที่จะเคลื่อนย้ายโปรแกรมระหว่างเครื่องคอมพิวเตอร์

วิธีการทางซอฟต์แวร์ก็มีข้อเสียที่ว่ามีประสิทธิภาพต่ำ เนื่องจากค่าโสหุ้ยของการควบคุมมีมากกว่าของทางฮาร์ดแวร์มาก แต่เพราะความสะดวกในการเขียนโปรแกรมและการกำหนดมาตรฐานทางภาษา คำสั่งพื้นฐานสำหรับการให้สิทธิพิเศษ (Mutual exclusion primitive) จึงนิยมใช้วิธีทางซอฟต์แวร์ โดยเฉพาะอย่างยิ่งวิธีทางซอฟต์แวร์ระดับต่ำ (ระดับระบบควบคุม) ซึ่งในวิธีนี้ทั้งภาษาที่ใช้เขียนโปรแกรมและตัวโปรแกรมระบบควบคุม (OS) จะถูกดัดแปลงเพิ่มเติมให้สามารถสนับสนุนการทำงานของคำสั่งพื้นฐานสำหรับการให้สิทธิพิเศษ สำหรับในบทความชุดนี้ผู้เขียนจะไม่อธิบายอย่างลึกซึ้งสำหรับวิธีการให้สิทธิพิเศษที่นิยมใช้กันอยู่ เพราะยุ่งยากและซับซ้อนมาก เพื่อให้เป็นการง่ายแก่ผู้อ่านในการอ่านโปรแกรมที่จะยกเป็นตัวอย่างต่อไป ส่วนที่เป็นแดนวิกฤตจะถูกล้อมรอบด้วยคำสั่ง ENTER CRITICAL SECTION และ END CRITICAL SECTION

1.4 การควบคุมให้โปรเซสทำงานอย่างสอดคล้องกัน

ปัญหาที่สำคัญอีกอย่างหนึ่งในการเขียนโปรแกรมเพื่อการคำนวณแบบ

ขนาน คือ การควบคุมให้โปรเซสทำงานอย่างสอดคล้องกัน เพราะในการคำนวณแบบขนานโปรเซสแต่ละตัวจะคำนวณอย่างอิสระไม่ขึ้นต่อกันบนโปรเซสเซอร์ที่ถูกกำหนดสำหรับโปรเซสนั้น ๆ แต่ระหว่างการคำนวณ โปรเซสหนึ่งอาจจำเป็นที่จะต้องใช้ผลลัพธ์หรือข้อมูลของอีกโปรเซสหนึ่ง ถ้าไม่มีการควบคุมให้การคำนวณเป็นไปอย่างสอดคล้องกัน โปรเซสที่ต้องการผลลัพธ์ของอีกโปรเซสหนึ่ง อาจจะไปอ่านเอาข้อมูลที่ไม่มีความหมายมาโดยคิดว่าเป็นผลลัพธ์ของโปรเซสที่ต้องการในขณะที่โปรเซสที่เป็นเจ้าของผลลัพธ์นั้นยังคำนวณไม่เสร็จ และยังไม่ได้เขียนผลลัพธ์ลงไปในหน่วยความจำส่วนกลาง

เทคนิคการควบคุมให้โปรเซสทำงานอย่างสอดคล้องกันนั้นมีอยู่หลายวิธี วิธีที่นิยมคือแนวกั้น (Barrier) และการหยุดรอ (Wait)

แนวกั้นคือ จุดในโปรแกรมที่โปรเซสทุกตัวต้องมาหยุดรอจนกระทั่งโปรเซสสุดท้ายเข้ามาถึง หลังจากนั้นทุกโปรเซสจะสามารถทำงานต่อไปได้ตามคำสั่งในโปรแกรมของแต่ละโปรเซส (คำสั่งของแต่ละโปรเซสอาจจะไม่เหมือนกัน แต่ที่มีเหมือนกันคือ คำสั่งแนวกั้นแทรกอยู่เพื่อให้แน่ใจว่า เมื่อถึงจุดนี้ต้องรอให้โปรเซสทั้งหมดทำงานมาถึงจุดที่ต้องการก่อน จึงจะเริ่มทำงานต่อไป) ข้อเสียของแนวกั้นคือ โปรเซสที่เข้ามาหยุดรอที่แนวกั้นนั้น อาจจะต้องเสียเวลารอนาน ถ้าโปรแกรมออกแบบมาไม่ดีพอ เป็นเหตุให้ประสิทธิภาพของระบบลดลง

การหยุดรอ คือ วิธีที่โปรเซสหนึ่ง ๆ มาหยุดรอสถานการณ์บางอย่างให้เกิดขึ้นก่อนที่จะเริ่มทำงานต่อไป เช่นรอให้อีกโปรเซสหนึ่งส่งสัญญาณมาบอกว่าพร้อมที่จะรับคำสั่งบางอย่าง ข้อแตกต่างระหว่างการหยุดรอกับแนวกั้นคือ ในเทคนิคการหยุดรอนั้น โปรเซสทุกตัวไม่จำเป็นต้องมาหยุดรอซึ่งกันและกัน เมื่อสถานการณ์ที่ต้องการเกิดขึ้นแล้ว โปรเซสที่หยุดรอก็สามารถทำงานต่อไปได้ทันที

2. ระบบส่งผ่านข้อมูล (Message passing)

การเขียนโปรแกรมเพื่อการคำนวณแบบขนานในระบบส่งผ่านข้อมูลนี้ง่ายกว่าของระบบหน่วยความจำร่วมมาก (แต่การออกแบบโปรแกรมให้มีประสิทธิภาพสูงนั้นยากกว่า) ภาษาที่ใช้เขียนโปรแกรมแทบจะไม่ต้องแก้ไขเป็นพิเศษจากภาษาสำหรับการคำนวณแบบอนุกรมเลย เพียงแค่เพิ่มคำสั่งพื้นฐานอีกสองคำสั่งในภาษามาตรฐาน คำสั่งดังกล่าวคือ SEND และ RECEIVE

SEND ใช้สำหรับส่งข้อมูลจากโปรเซสหนึ่งไปยังโปรเซสอื่น ๆ การส่งข้อมูลเป็นไปได้สองรูปแบบคือ ส่งข้อมูลไปยังโปรเซสใดโปรเซสหนึ่งอย่างเจาะจง หรือส่งข้อมูลไปยังทุกโปรเซสที่มีอยู่ในระบบ (ส่งแบบกระจายเสียง) รายละเอียดของข้อมูลที่จะส่ง อย่างน้อยจะต้องรวมถึงความยาวของข้อมูล จุดหมายของข้อมูล และข้อมูลที่ต้องการส่งจริง

RECEIVE ใช้สำหรับรับข้อมูลที่ส่งมาจากโปรเซสอื่น เช่นเดียวกับการส่งการรับข้อมูล มีอยู่สองรูปแบบคือ รับข้อมูลที่ส่งมาจากโปรเซสใดโปรเซสหนึ่งโดยเฉพาะ หรือรับข้อมูลจากโปรเซสใดก็ได้ที่ส่งมาถึงโปรเซสนั้นก่อน

ตัวอย่างง่าย ๆ

วิธีที่ดีที่สุด เพื่อช่วยให้ผู้อ่านเข้าใจเครื่องมือในการเขียนโปรแกรมเพื่อการคำนวณแบบขนานที่ได้อธิบายมาทั้งหมด ผู้เขียนจะขอยกตัวอย่างง่าย ๆ สักอัน ตัวอย่างนี้คือ การหาผลบวกของเลขหนึ่งล้านตัว ที่จริงปัญหานี้ไม่ง่ายนักทีเดียวถ้าจะให้โปรแกรมคำนวณได้อย่างมีประสิทธิภาพ แต่ในที่นี้ผู้เขียนจะแสดงเฉพาะวิธีง่าย ๆ เพื่อให้ผู้อ่านทำความเข้าใจกับการเขียนโปรแกรมเพื่อการคำนวณแบบขนานได้สะดวกขึ้น

```
 1 program     serial;
 2 const       N=1000000;
 3 var         A:array [1...N] of integer;
 4             result:integer;
 5             i:integer;
 6
 7 procedure   SUM (first, last:integer);
 8       begin
 9             for i := first to last do
10             result := result + A [i];
11       end;
12
13 begin
14       for i := 1 to N do
15          read (A[i]);
16       result := 0;
17       call SUM (1, 1000000);
18       write (result);
19 end.
```

รูปที่ ๑ โปรแกรมสำหรับหาค่าผลบวกของเลขหนึ่งล้านตัว ที่ใช้คำนวณบนเครื่องคอมพิวเตอร์แบบโปรเซสเซอร์เดี่ยว

ตัวอย่างในรูปที่ ๑ เป็นโปรแกรมที่ใช้คำนวณบนเครื่องคอมพิวเตอร์แบบโปรเซสเซอร์เดี่ยว โปรแกรมหลักทำหน้าที่อ่านตัวเลขที่ต้องการบวกทั้งหมดเข้ามาเก็บไว้ใน array A หลังจากนั้นโปรแกรมหลักจะเรียกโปรแกรมย่อย SUM มาคำนวณหาค่าผลบวกของเลขล้านตัวที่เก็บอยู่ใน array A เมื่อเสร็จการบวกแล้ว โปรแกรมหลักจะสั่งพิมพ์ผลลัพธ์ของการคำนวณออกมา

สำหรับการเขียนโปรแกรมเพื่อการคำนวณแบบขนาน เราจำเป็นที่จะต้องรู้ว่าในระบบคอมพิวเตอร์ที่ใช้งานอยู่มีกี่โปรเซสเซอร์ เพื่อที่เราจะได้แบ่งงานให้แต่ละโปรเซสเซอร์ทำงานได้

เท่ากัน ในปัจจุบันเครื่องบางรุ่นมีหน่วยความจำหรือตัวแปรพิเศษที่ใช้เก็บค่านี้อยู่ แต่เพื่อให้ง่ายต่อความเข้าใจในตัวอย่างที่จะเสนอต่อไป เราจะสมมติให้ระบบคอมพิวเตอร์ที่ใช้งานมีโปรเซสเซอร์อยู่ 5 ตัว

ระบบหน่วยความจำร่วม : ตัวอย่างโปรแกรมในรูปที่ 10 เป็นโปรแกรมที่ใช้สำหรับบวกเลขหนึ่งล้านตัวสำหรับเครื่องแบบหลายโปรเซสเซอร์ระบบหน่วยความจำร่วม สิ่งแรกที่สุดก็คือการกำหนดข้อมูลส่วนกลางและข้อมูลส่วนตัว array A ในบรรทัดที่ 3 และตัวแปร result ในบรรทัดที่ 4 ถูกกำหนดให้เป็นข้อมูลส่วนกลาง ตัวแปรที่นอกเหนือจากสองตัวข้างต้นจะเป็นข้อมูลส่วนตัว ดังนั้นแต่ละโปรเซส (ระหว่างการคำนวณแบบขนาน) จะมีข้อมูลของตัวแปรเหล่านี้โปรเซสละหนึ่งชุด ซึ่งค่าของตัวแปรเหล่านี้ในแต่ละโปรเซสไม่จำเป็นต้องเป็นค่าเดียวกันและไม่มีความสัมพันธ์ซึ่งกันและกันเลย

```
1  program shared_memory;
2  const     N = 1000000;
3  shared    A      : array [1...N] of integer;
4            result : integer;
5  var       i      : integer;
6
7  procedure    SUM (first, last : integer);
8            var    x : integer;
9            begin
10                  for i := first to last do
11                  begin
12             BEGIN CRITICAL SECTION
13                  result := result + A [i];
14             END CRITICAL SECTION
15                  end;
16           end;
17
18 begin
19           for i := 1 to N do
20                  read (A[i]);
21           result := 0;
22 PARBEGIN
23           call SUM (1, 200000);
24           call SUM (200001, 400000);
25           call SUM (400001, 600000);
26           call SUM (600001, 800000);
27           call SUM (800001, 1000000);
28 PAREND
29 write (result);
30 end.
```

รูปที่ 10 โปรแกรมสำหรับหาค่าผลบวกของเลขหนึ่งล้านตัวที่ใช้คำนวณบนเครื่องคอมพิวเตอร์แบบหลายโปรเซสเซอร์ระบบหน่วยความจำร่วม

ระหว่างบรรทัดที่ 19 ถึง 21 จะเป็นการคำนวณแบบอนุกรม โดยโปรเซสเซอร์เพียงตัวเดียว ซึ่งช่วงนี้จะเป็นการอ่านตัวเลขทั้งล้านตัวเข้ามาเก็บไว้ใน array A ซึ่งเป็นข้อมูลส่วนกลาง ในบรรทัดที่ 23 ถึง 27 จะเป็นช่วงการคำนวณแบบขนานโดยโปรเซสเซอร์ 5 ตัว เพื่อเป็นการแบ่งงานให้โปรเซสเซอร์แต่ละตัวทำงานเท่า ๆ กัน เราจะแบ่งให้แต่ละโปรเซสที่คำนวณอยู่บนแต่ละโปรเซสเซอร์หาค่าผลบวกของเลขจำนวนสองแสนตัวจาก array A โดยที่แต่ละโปรเซสจะบวกเลขคนละชุดกัน ในโปรแกรมย่อย SUM เราจะเห็นได้ว่าทุกครั้งที่มีการบวก (บรรทัดที่ 12 ถึง 14) โปรเซสจะเข้าสู่แดนวิกฤต เนื่องจากมีการเปลี่ยนแปลงค่าของข้อมูลส่วนกลาง (result) ข้อเสียของจุดนี้คือโปรแกรมจะทำงานได้ช้าลง เพราะทุกโปรเซสต้องรอเข้าสู่แดนวิกฤตตามลำดับ ผลลัพธ์ที่ตามมาคือ เราสูญเสียความสามารถในการคำนวณแบบขนานไป เพราะในเวลาหนึ่ง ๆ จะมีเพียงแค่หนึ่งโปรเซสเท่านั้นที่ได้รับอนุญาตให้เข้าไปแก้ไขค่าของ result ในแดนวิกฤต

เราสามารถแก้ไขข้อเสียของโปรแกรมที่ว่านี้ได้โดยพยายามให้โปรเซสเข้าสู่แดนวิกฤตให้น้อยที่สุด ในตัวอย่างรูปที่ 11 เราจะเห็นได้ว่า ระหว่างการคำนวณผลบวกของแต่ละโปรเซสอยู่ เราใช้ตัวแปรส่วนตัวในการคำนวณ ทำให้ไม่จำเป็นต้องเข้าสู่แดนวิกฤตทุกครั้งที่บวก (สำหรับการใช้ค่าใน array A นั้น เราไม่ต้องเข้าสู่แดนวิกฤต เนื่องจากโปรแกรมของเราไม่มีการเปลี่ยนค่าใน array A เลย ดังนั้น โปรเซสแต่ละตัวเข้าไปอ่านค่าได้พร้อม ๆ กัน) หลังจากที่ผลบวกย่อย (X) ของเลขสองแสนตัว ได้ถูกคำนวณเรียบร้อยแล้ว โปรเซสแต่ละตัวจะเอาผลบวกย่อยนี้ไปบวกใส่ผลลัพธ์รวม (result) ดังนั้น แต่ละโปรเซสจะเข้าสู่แดนวิกฤตเพียงครั้งเดียว

```
31 procedure  SUM (first, last : integer);
32 var        x : integer;
33 begin
34           x := 0;
35           for i := first to last do
36                x := x + A [i];
37           BEGIN CRITICAL SECTION
38           result := result + x;
39           END CRITICAL SECTION
40 end;
```

รูปที่ 11 โปรแกรมย่อย SUM ที่ได้รับการปรับปรุงให้การเข้าสู่แดนวิกฤตมีน้อยที่สุด

ระบบส่งผ่านข้อมูล : สไตล์การเขียนโปรแกรมในระบบนี้ค่อนข้างจะแตกต่างจากระบบหน่วยความจำร่วม สาเหตุแรกที่ทำให้เกิดความแตกต่างคือ ข้อมูลที่จำเป็นสำหรับการคำนวณจะต้องส่งจากหน่วยความจำของโปรเซสเซอร์หนึ่งไปยังอีกโปรเซสเซอร์หนึ่ง สาเหตุที่สองคือ ในระบบส่งผ่านข้อมูลนี้จะไม่มีโปรแกรมหลักทำหน้าที่เป็นเจ้านายคอยควบคุมการทำงาน และคอยกำหนดว่าส่วนไหนจะคำนวณแบบอนุกรมหรือแบบขนาน เนื่องจากการสั่งงานดังกล่าวจำเป็นต้องมีการส่งส่วนของโปรแกรมไปมาระหว่างโปรเซสเซอร์ ซึ่งค่าโสหุ้ยในการส่งโปรแกรมนี้มีราคาค่อนข้างสูง มีผลให้ประสิทธิภาพโดยส่วนรวมลดลง ดังนั้น ในระบบส่งผ่านข้อมูลนี้ ตอนเริ่มต้นโปรแกรมจะถูกโหลดไปยังโปรเซสเซอร์แต่ละตัวโดยตรง

รูปที่ 12 แสดงรายละเอียดของโปรแกรมที่คำนวณอยู่บนโปรเซสเซอร์หมายเลข 1 ส่วนรูปที่ 13 เป็นรายละเอียดของโปรแกรมที่คำนวณอยู่บนโปรเซสเซอร์หมายเลข 2 ถึง 5 การทำงานจะเริ่มต้น

โดย โปรเซส 1 อ่านตัวเลขหนึ่งล้านตัวเข้ามาเก็บไว้ใน array A ซึ่งเป็นตัวแปรส่วนตัวของโปรเซส 1 ในระหว่างนี้ โปรเซสอื่น ๆ จะหยุดรอที่คำสั่ง RECEIVE เนื่องจากยังไม่มีข้อมูลส่งมา ถึง หลังจากโปรเซส 1 อ่านข้อมูลเสร็จก็จะเริ่มแบ่งงานโดยจะส่งเลขล้านตัวนี้ไปให้โปรเซสอื่นช่วยกันบวกโปรเซสละสองแสนตัว การส่งทำโดยคำสั่ง SEND ในบรรทัดที่ 18 ถึง 21 จากบรรทัดที่ 18 เราสามารถตีความได้ว่า ให้ส่งข้อมูลไปยังโปรเซสเซอร์ 2 ข้อมูลที่ส่งมีความยาวสองแสนคำ โดยข้อมูลที่ส่งเริ่มต้นที่ array A [1] หลังจากแบ่งงานให้โปรเซสอื่น ๆ ทั้งหมดแล้วโปรเซส 1 จะเหลืองานอีกส่วนหนึ่งไว้คำนวณเองในบรรทัดที่ 22 ไม่เช่นนั้นระหว่างที่โปรเซส 1 รอให้โปรเซสอื่นคำนวณอยู่

```
1  program   message_passing1;
2  const       N = 1000000;
3  var         A : array [1...N] of integer;
4            result : integer;
5               i : integer;
6            temp : integer;
7
8  procedure    SUM (first, last:integer);
9          begin
10                for i := first to last do
11                 result := result + A [i];
12         end;
13
14 begin
15         for i := 1 to N do
16                read (A [i]);
17         result := 0;
18         SEND (2, 200000, A [1]);
19         SEND (3, 200000, A [200001]);
20         SEND (4, 200000, A[400001];
21         SEND (5, 200000, A[600001];
22         call SUM (800001, 1000000);
23         for i := 2 to 5 do
24         begin
25         RECEIVE FROM ANY (temp);
26         result := result + temp;
27         end;
28         write (result);
29 end.
```

รูปที่ 12 โปรแกรมสำหรับหาค่าผลบวกของเลขหนึ่งล้านตัวที่ใช้คำนวณบนโปรเซสเซอร์ตัวที่ 1 ของระบบส่งผ่านข้อมูล

ตัวเองอาจจะไม่มีงานทำ

สำหรับโปรเซส 2 ถึง 5 เมื่อได้รับข้อมูลที่ส่งมาจากโปรเซส 1 ในบรรทัดที่ 44 แล้ว (ข้อมูลที่รับมาจะถูกเก็บอยู่ในตัวแปรส่วนตัว A) แต่ละโปรเซสจะเริ่มบวกเลขที่ตัวเองได้รับจนเสร็จ หลังจากที่บวกเสร็จก็จะส่งผลลัพธ์ย่อยนี้กลับไปให้โปรเซส 1 เพื่อบวกผลลัพธ์ย่อยนี้เข้าด้วยกัน เพื่อจะได้เป็นผลลัพธ์ของโปรแกรมอีกทอดหนึ่ง โปรแกรมระหว่างบรรทัดที่ 23 ถึง 27 จะเป็นส่วนที่โปรเซส 1 มารอรับผลลัพธ์ย่อยจากโปรเซสอื่นทีละหนึ่งตัว บวกผลลัพธ์ย่อยนี้เข้าเป็นผลลัพธ์จริง เมื่อวนจนครบทุกโปรเซสแล้ว โปรเซส 1 จะพิมพ์ผลลัพธ์ออกมา

จากตัวอย่างที่ยกมา เราจะเห็นได้ว่า**การเขียนโปรแกรม เพื่อการคำนวณแบบขนานสำหรับเครื่องที่มีระบบภายในต่างกัน จะมีสไตล์การเขียนต่างกันอย่างเห็นได้ชัด** แต่ละระบบจะมีข้อดีข้อเสียของตัวเอง ยกตัวอย่างเช่น **ระบบหน่วยความจำร่วม การออกแบบอัลกอริทึมของโปรแกรมจะค่อนข้างง่าย** เมื่อเทียบกับระบบส่งผ่านข้อมูล เนื่องจากเรามีหน่วยความจำร่วม ดังนั้น ข้อมูลไหนที่ต้องการใช้ร่วมกันก็เพียงแต่กำหนดให้เป็นตัวแปรร่วมก็เป็นอันใช้ได้ แต่ข้อเสียคือ หากมีข้อผิดพลาดในโปรแกรม หรือเรียกง่าย ๆ ว่า บั๊ค เราจะค้นหา**บั๊คได้ลำบากมาก** เพราะเราไม่รู้ว่าโปรเซสใดเป็นตัวก่อเหตุ หรือถ้าโปรแกรมทำงานไม่สอดคล้องกัน ก็อาจจะหยิบเอาค่าที่ไม่มีความหมายไปใช้ ซึ่งยากที่จะหาที่ผิดนี้ได้

สำหรับ**ระบบส่งผ่านข้อมูล** ข้อดีข้อเสียเกือบจะเป็นไปในทางตรงกันข้าม คือ **การออกแบบอัลกอริทึมค่อนข้างจะยุ่งยาก** เพราะเราจำเป็นต้องออกแบบให้โปรแกรมส่งข้อมูลข้ามไปมาระหว่างโปรเซสให้น้อยที่สุด ไม่เช่นนั้นประสิทธิภาพของระบบจะลดลง แต่การเขียนโปรแกรมและการหาที่ผิดพลาดจะง่ายกว่าของระบบหน่วยความจำร่วมมาก เพราะการทำงานของแต่ละโปรเซสแทบไม่เกี่ยวข้องกันเลย **เมื่อมีบั๊คเกิดขึ้น เราสามารถจะค้นหาได้ที่โปรเซสต้นตอโดยตรง**

```
30 program     message_passingN;
31 const       N = 200000;
32 var         A:array [1...N] of integer;
33             result:integer;
34             i:integer;
35
36 procedure   SUM (first, last:integer);
37         begin
38                for i := first to last do
39                result := result + A [i];
40         end;
41
42 begin
43         result := 0;
44         RECEIVE (1, N, A [i]);
45         call SUM (1, 200000);
46         SEND (1, 1, result);
47 end.
```

รูปที่ 13 โปรแกรมสำหรับหาค่าผลบวกของเลขล้านตัวที่ใช้คำนวณบนโปรเซสเซอร์ที่ 2 ถึง 5 ของระบบส่งผ่านข้อมูล

สำหรับในระบบร่วม (Hybrid) **ในแง่ของการเขียนโปรแกรมนับว่าแย่ที่สุดในบรรดาระบบต่าง ๆ** การค้นหาบั๊คค่อนข้างยากเพราะจะเป็นบั๊คประเภทเดียวกับระบบหน่วยความจำร่วม การออกแบบอัลกอริทึมจำเป็นจะต้องพิจารณาการวางตำแหน่งของข้อมูลในหน่วยความจำเช่นเดียวกับระบบส่งผ่านข้อมูล แต่ข้อดีของระบบนี้ที่ยังทำให้ได้รับการสนใจค้นคว้าอย่างแพร่หลายอยู่คือ การออกแบบสร้างเครื่องระบบนี้ทำได้ง่ายที่สุด

ในฉบับหน้า ซึ่งเป็นตอนจบ ผู้เขียนจะพูดถึงการเขียนโปรแกรมเพื่อหาตัวเลขพราม (prime number) อย่างมีประสิทธิภาพ พร้อมทั้งยกตัวอย่างอัลกอริทึมของโปรแกรมในการหาตัวเลขพรามนี้ สำหรับเครื่องแบบหลายโปรเซสเซอร์